SQL 语句允许将一个查询语句做为一个结果集供其他 SQL 语句使用，就像使用普通的表一样，被当作结果集的查询语句被称为子查询。  
所有可以使用表的地方几乎都可以使用子查询来代替。  
可以将子查询看成一张暂态的数据表，这张表在查询开始时被创造，在查询结束时被删除。  
如：SELECT \* FROM(SELECT \* FROM T2 where FAge<30)

**1.1、单值子查询**  
单值子查询的的返回值必须只有一行记录，而且只能有一个列。  
这样的子查询又被称为标量子查询。  
例：  
SELECT 1 AS f1,2,(SELECT MIN(FYearPublished) FROM T\_Book),(SELECT MAX(FYearPublished) FROM T\_Book) AS f4;

如果一个子查询返回值不止一行记录或者有多个列的话都不能当作标量子查询使用，否则会出错。有的时候一个子查询是否使用正确是要到运行时才能确定的，比如某条件语句查询到的结果。  
当子查询跟随在 =、!=、<、<=、>、>= 之后，或子查询用作表达式时，这种情况是不允许的。

**1.2、列值子查询**  
列值子查询可以返回一个多行多列的结果集。这样的子查询又被称为表子查询，表子查询可以看作一个临时的表。

在 FROM 子句中使用的最简单的表子查询：  
SELECT T\_Reader.FName,t2.FYearPublished,t2.FName  
FROM T\_Reader,  
(SELECT \* FROM T\_Book WHERE FYearPublished < 1800) t2  
这里将 T\_Reader 表和表子查询做交叉连接，并且将 “SELECT \* FROM T\_Book WHERE FYearPublished < 1800” 做为表子查询，还可以为表子查询执行表别名，在 SELECT  
的列表中也可以使用和表一样的列名引用方式，这与使用一个普通的数据表没有什么区别。

表子查询可以看作一张临时的表，所以引用子查询中列的时候必须使用子查询中定义的列名，也就是如果子查询中为列定义了别名，那么在引用的时候也要使用别名。  
如：  
SELECT T\_Reader.FName,t2.FYear,t2.FName ,t2.F3  
FROM T\_Reader,  
(SELECT FYearPublished AS FYear,FName,1+2 as F3 FROM T\_Book WHERE  
FYearPublished < 1800) t2  
这里的表子查询为 FYearPublished 列取了一个别名 FYear，这样在引用它的时候就必须使用 FYear 而不能继续使用 FYearPublished 这个名字，这里子查询中还增加了一个新列 F3，同样可以在 SELECT 列表中引用它。

**2、SELECT 列表中的标量子查询**  
标量子查询完全可以返回随当前查询记录而变化的值。  
如：  
SELECT FId,FName,  
(  
SELECT MAX(FYearPublished)  
FROM T\_Book  
WHERE T\_Book. FCategoryId= T\_Category.FId  
)  
FROM T\_Category  
这个 SELECT 语句首先检索 FId、FName 两个字段，而第三个字段不是一个列二是一个子查询。这个子查询位于主查询的内部，它返回一类图书的最新出版年份。因为聚合函数仅返回一行记录，所以这满足标量子查询的条件。通过 WHERE 语句，这个子查询也被连接到外部的 SELECT 查询语句中，因为这个连接，MAX(FYearPublished) 将返回每类图书的最新出版年份。  
这里的子查询是依赖于外部查询中的 T\_Category.FId 字段的，这个子查询是无法单独执行的。

相关子查询：引用了外部查询中字段的子查询。

上个 sql 语句中如果没有子查询中的 WHERE 子句将子查询连接到外部查询，则结果将只是子查询返回的所有记录的最大值。  
如：将子查询中的WHERE 子句删除：  
SELECT FId,FName,  
(  
SELECT MAX(FYearPublished)  
FROM T\_Book  
)  
FROM T\_Category  
MAX(FYearPublished) 现在是 T\_Book 表中所有记录的最大出版年份，它不与任何书籍分类相关。

**三、WHERE 子句中的标量子查询**

检索喜欢 “Story” 的读者主键列表：  
SELECT FReaderId FROM T\_ReaderFavorite  
WHERE FCategoryId=  
(  
SELECT FId FROM T\_Category  
WHERE FName=’Story’  
)

检索每一种书籍类别中出版年份最早的书籍的名称，如果有两本或者多本书籍在同一年出版，则均显示它们的名字。要求检索结果中显示出类型的名字、书的名字和它的出版年份：  
SELECT T\_Category.FId, T\_Book. FName,T\_Book.FYearPublished  
FROM T\_Category  
INNER JOIN T\_Book ON T\_Category.FId=T\_Book.FCategoryId  
WHERE T\_Book.FYearPublished=  
(  
SELECT MIN(T\_Book.FYearPublished)  
FROM T\_Book  
WHERE T\_Book.FCategoryId=T\_Category.FId  
)  
T\_Category 表和 T\_Book 表首先进行内部连接，然后使用 WHERE 子句中使用子查询来进行数据的过滤。这个子查询是一个相关子查询，它返回外部查询中当前图书类别中的图书的最早出版年份。在外部查询的WHERE子句中，T\_Book 的 FYearPublished 与子查询的返回值进行比较，这样就可以得到每种书籍类型中的出版最早的书籍了。  
注：  
所有在 SELECT 列表中的字段如果没有包含在聚合函数中，则必须放到 GROUP BY 子句中，所以将T\_Book. FName 加入到 GROUP BY 子句中。

**四、集合运算符与子查询**

4.1 IN 运算符  
使用 IN 运算符可以用来匹配一个固定集合中的某一项。  
如：检索在2001、2003 和2005年出版的所有图书：  
SELECT \* FROM T\_Book  
WHERE FYearPublished IN(2001,2003,2005)

检索所有图书出版年份内入会的读者信息:  
SELECT \* FROM T\_Reader  
WHERE FYearOfJoin IN  
(  
select FYearPublished FROM T\_Book  
)

4.2 ANY 和 SOME 运算符  
在 SQL 中 ANY 和 SOME 是同义词，SOME 的用法和功能和 ANY一模一样。  
ANY 必须和其他的比较运算符共同使用，而且必须将比较运算符放在 ANY 关键字之前，所比较的值需要匹配子查询中的任意一个值。

检索所有图书出版年份内入会的读者信息：  
SELECT \* FROM T\_Reader  
WHERE FYearOfJoin =ANY  
(  
select FYearPublished FROM T\_Book  
)  
外部查询中的 WHERE 子句指定 FYearOfJoin 必须等于子查询 select FYearPublished FROM T\_Book所返回的集合中的任意一个值。

检索所有图书出版年份内入会的读者信息:  
SELECT \* FROM T\_Reader  
WHERE FYearOfJoin IN  
(  
select FYearPublished FROM T\_Book  
)

注：  
“=ANY” 等价于 IN 运算符  
“<>ANY” 等价于 NOT IN 运算符。

检索在任何一个会员出生之前出版的图书：  
SELECT \* FROM T\_Book  
WHERE FYearPublished<any (="" select="" fyearofbirth="" from="" t\_reader="" )="" any="" 运算符不能与固定的集合相匹配，因为没有对固定的集合进行="" 匹配的必要，待匹配的集合是固定的：="" \*="" t\_book="" where="" fyearpublished<2005="" 4.3="" all运算符="" all="" 运算符要求比较的值需要匹配子查询中的所有值。="" 运算符同样不能单独使用，必须和比较运算符共同使用。="" 检索在所有会员入会之前出版的图书：="" fyearpublished<all="" fyearofjoin="" 运算符同样不能与固定的集合相匹配，比如下面的sql="" 语句是错误的：="" fyearpublished<all(2001,2003,2005)="" 因为没有对固定的集合进行all="" 匹配的必要，待匹配的集合是固定的，="" 所以上面的sql语句完全可以用下面的sql语句来代替：="" fyearpublished<2001="" 当使用="" 运算符的时候，如果带匹配的集合为空，也就是子查询没有返回任何数据的时候，不论与什么比较运算符搭配使用="" 的返回值将永远是="" true。="" 如：="" 下面的sql语句用于检索在所有江苏省会员入会之前出版的图书：="" fprovince="JiangSu" 这个查询结果将所有的会员都检索出来了，可是根本没有江苏省的会员，应该是返回空结果才对的。其实这完全符合="" 运算符的语义，因为没有江苏省的会员，所以每本书的出版年份就在所有的江苏省的会员之前，所以每一本书都符合匹配条件。="" 4.4="" exists="" 运算符="" 运算符是单目运算符，它不与列匹配，因此它也不要求待匹配的集合是单列的。="" 运算符用来检查每一行是否匹配子查询，可以认为="" 就是用来测试子查询的结果是否为空，如果结果集为空则匹配结果为="" false，否则匹配结果为="" 对="" 表中的每行数据进行匹配，测试是否存在山东省的读者，因为系统中存在山东省的读者，所以这个sql语句将检索出所有的图书。="" 表中的每行数据进行匹配，测试是否存在云南省的读者，因为系统中不存在云南省的读者，所以这个="" sql="" 语句的执行结果为空。="" 运算符的真正意义只有和相关子查询一起使用。="" 相关子查询中引用外部查询中的这个字段，这样在匹配外部子查询中的每行数据的时候相关子查询就="" 会根据当前行的信息来进行匹配判断。="" 检索存在="" 1950="" 年以前出版的图书的图书类别：="" t\_category="" t\_book.="" fcategoryid="T\_Category.FId" and="" fyearpublished<1950="" 在="" 后的子查询中，sql="" 表中的每一行数据到子查询中进行匹配，测试="" 表中是否存在="" 字段值等于当前类别主键值且出版年份在="" 年之前的书籍。="" **五、在其他类型SQL语句中的子查询应用**

5.1 子查询在 INSERT 语句中的应用  
在使用 INSERT 语句的时候，一般都是使用它向数据库中一条条的插入数据，比如：  
INSERT INTO MyTable(FId,FName,FAge)  
VALUES(1,’John’,20)  
但是有时可能需要将数据批量插入表中，比如创建一个和 T\_ReaderFavorite 表结构完全相同的表 T\_ReaderFavorite2，然后将 T\_ReaderFavorite 中的输入复制插入到 T\_ReaderFavorite2 表。  
首先创建 T\_ReaderFavorite2 表：  
CREATE TABLE T\_ReaderFavorite2 (FCategoryId INT,FReaderId INT)

除了 INSERT……VALUES…… 这种用法外，INSERT 语句还支持另外一种语法：INSERT……SELECT……，采用这种使用方式可以将 SELECT 语句返回的结果集直接插入到目标表中，因为这一切都是都数据库内部完成的，所以效率非常高。

使用 INSERT……SELECT…… 来实现将 T\_ReaderFavorite 中的输入复制插入到 T\_ReaderFavorite2 表：  
INSERT INTO T\_ReaderFavorite2(FCategoryId,FReaderId)  
SELECT FCategoryId,FReaderId FROM T\_ReaderFavorite  
这里使用 SELECT FCategoryId, FReaderId FROM T\_ReaderFavorite将 T\_ReaderFavorite 表中的数据读出，然后使用 INSERT INTO T\_ReaderFavorite2(FCategoryId,FReaderId) 将检索结果插入到T\_ReaderFavorite2 表中，注意上下的列顺序必须是一一对应的。

使用 INSERT……SELECT…… 不仅能够实现简单的将一个表中的数据导出到另外一个表中的功能，还能在将输入插入目标表之前对数据进行处理。  
如：  
将 T\_ReaderFavorite 表中的数据复制到 T\_ReaderFavorite2 表中，但是如果 T\_ReaderFavorite 表中的 FReaderId 列的值大于 10，则将 FReaderId 的值减去 FCategoryId 的值后再复制到T\_ReaderFavorite2 表中：  
INSERT INTO T\_ReaderFavorite2(FCategoryId,FReaderId)  
SELECT FCategoryId,  
(CASE  
WHEN FReaderId<=10 THEN FReaderId ELSE FReaderId- FCategoryId END ) FROM T\_ReaderFavorite 这里在 SELECT 语句中使用 CASE 函数来实现对数据插入前的处理。 使用这种插入前的数据处理可以完成诸如 “将数据从 A 表导出到 B 表，并且将 B 表的主键全部加上bak 前缀”、 “将A 公司的所有员工插入到我们的会员表，自动导入所有的客户信息，并且为其自动生成会员编号” 等复杂的任务。 因为可以在插入目标表前可以对数据进行处理，所以 INSERT……SELECT…… 语句不局限于同结构表间的数据插入，也可以实现异构表见输入的插入。 假设要将所有会员爱好的图书统一增加 “小说”，也就是为 T\_Reader 表中的每个读者都在T\_ReaderFavorite 表中创建一条 FCategoryId 等于 1 的记录： INSERT INTO T\_ReaderFavorite(FCategoryId,FReaderId) SELECT 1,FId FROM T\_Reader WHERE NOT EXISTS ( SELECT \* FROM T\_ReaderFavorite WHERE T\_ReaderFavorite. FCategoryId=1 AND T\_ReaderFavorite. FReaderId= T\_Reader.FId ) SELECT 语句从 T\_Reader 表中检索所有的读者信息，并且将第一列设定为固定值 1，而将第二列设定为读者的主键。 **5.2 子查询在 UPDATE 语句中的应用**  
UPDATE 语句中可以在更新列表中以及 WHERE 语句使用子查询。

将图书的出版日期全部更新为所有图书中的最新出版日期：  
UPDATE T\_Book  
SET FYearPublished=  
(SELECT MAX(FYearPublished) FROM T\_Book)  
注：在MYSQL 中是不支持使用子查询来更新一个列的，所以这个 UPDATE 语句无法在 MYSQL 中执行。

如果 UPDATE 语句拥有 WHERE 子句，那么还可以在 WHERE 子句中使用子查询，其使用方式与 SELECT 语句中的子查询基本相同，而且也可以使用相关子查询等高级的特性。  
将所有同类书本书超过3 本的图书的出版日期更新为2005：  
UPDATE T\_Book b1  
SET b1.FYearPublished=2005  
WHERE  
(  
SELECT COUNT(\*) FROM T\_Book b2  
WHERE b1. FCategoryId=b2. FCategoryId  
)>3  
使用相关子查询来查询所有与待更新的书籍属于同类别的书籍的总数，如果总数大于 3 则将当前书籍的出版日期更新为 2005。

5.3 子查询在 DELETE 语句中的应用

子查询在 DELETE 中唯一可以应用的位置就是 WHERE 子句，使用子查询可以完成复杂的数据删除控制。其使用方式与 SELECT 语句中的子查询基本相同，而且也可以使用相关子查询等高级的特性。  
使用相关子查询来查询所有与待更新的书籍属于同类别的书籍的总数，如果总数大于 3 则将当前书籍删除：  
DELETE FROM T\_Book b1  
WHERE  
(  
SELECT COUNT(\*) FROM T\_Book b2  
WHERE b1. FCategoryId=b2. FCategoryId  
)